**Publish Date: April 1, 2014**

**Version: 2.4**

# Insertion of Metadata and LTC into Transport Stream

The purpose of this document is to document technical and feature specifications of inserting metadata and LTC packets into transport stream.


www.evertz.com

**EVERTZ MICROSYSTEMS LTD.**

***BURLINGTON, ONTARIO, CANADA***

# Section A

# Metadata Insertion

The purpose of this section is to outline the functionality of the Metadata insertion on 3080MUX-ASI32-IPGE-DMXE hardware.

www.evertz.com

**EVERTZ MICROSYSTEMS LTD.**
***BURLINGTON, ONTARIO, CANADA***

# 1.0 Purpose

The purpose of this section is to outline the requirements for providing the 3080MUX-ASI32-IPGE-DMXE (herein referred to as MUX) with metadata content that it can then transmit as part of the output transport stream service in the form of a transport stream packet.

# 2.0 Requirements

In order to facilitate the transfer of data to the MUX, user will require a local PC connected to the frame controller of the frame housing the MUX. Application code will need to run from that PC that can open a TCP socket at an agreed upon port, and transfer the required metadata over the socket as per the file formatting discussed in this document. See Figure 1: 3080MUX-ASI32-IPGE-DMXE System with Metadata Insertion

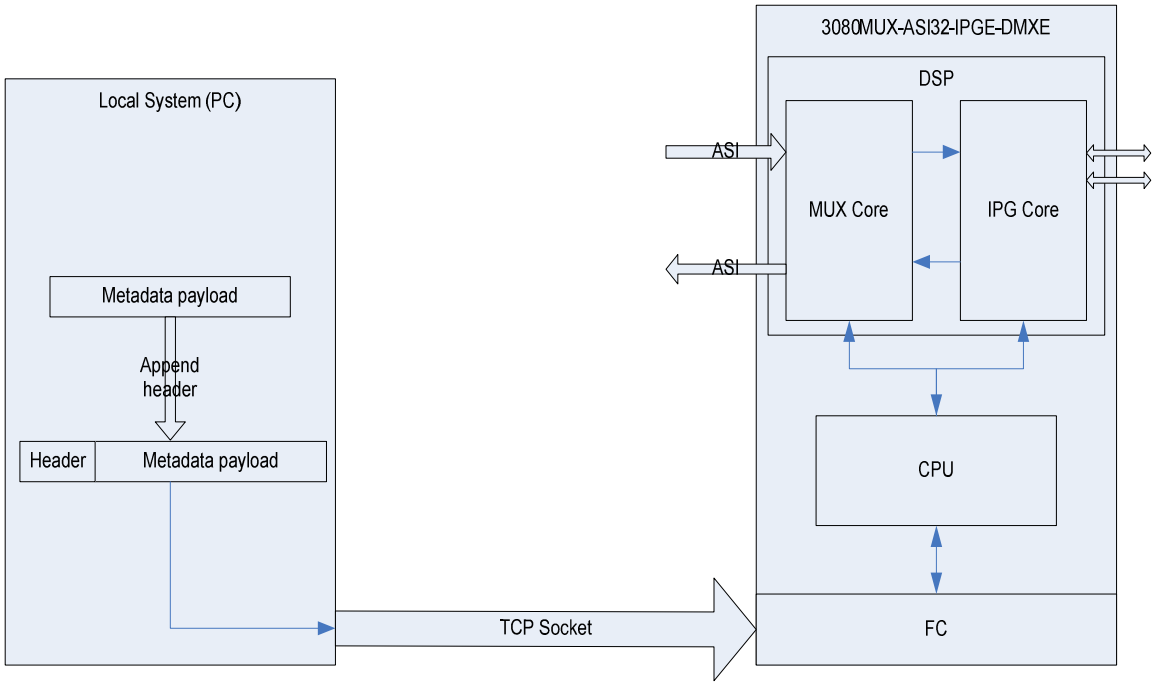



**Figure 1: 3080MUX-ASI32-IPGE-DMXE System with Metadata Insertion**

## 3.0 Metadata Payload Structure.

In order for the MUX to properly handle the metadata received, the metadata must conform to following structure.

**Metadata Header: 64 bytes**

| | |
|---|---|
| Header Version | 1 byte |
| Header Length | 1 byte |
| Program Reference | 1 byte |
| Association type | 1 byte |
| Association source | 16 bytes |
| Association program | 2 bytes |
| Association PID | 2 bytes |
| Inject time | 5 bytes |
| Payload length | 4 bytes |
| PTS Mode | 1 byte |
| PTS Offset | 4 bytes |
| Reserved | 26 bytes |

Metadata Payload: “Payload length” bytes

In order for the metadata payload to contribute correctly to the desired transport stream service, the header fields must contain relevant information for TS packet insertion. Note for header fields greater than 1 byte, most significant byte must be transmitted first.

## 3.1 Details about the Metadata Header fields

**Header Version**
Set value as 1.

**Header Length**
Value should always be 64.

**Program Reference**
Indicates whether the packet payload is contributing to an input or an output program
0 – input
1 – output
Other values will result in packet being discarded

**Association Type:**
Specifies the type of stream the “Association Source” refers to.
0 – ASI
1 – IP Stream Index
2 – IP Address
Others values will result in packet being discarded

**Association Source:**
Specifies the source of the association, dependant on the "Association Type"

If the association type is ASI, then the association source would be the ASI input or output number.

If the association type is IP Stream Index, then the association source would be the index of the stream on a particular IP input or output. See Figure 2: IP Stream Index Example.

If the association type is IP Address, then the association source will be the IP address of the stream, followed by the IP port number. In this mode, association source is structured as follows:

| | |
|---|---|
| Reserved | 10 bytes |
| IP Address | 4 bytes |
| Port | 2 bytes |

For example, if the specified program is being broadcast at multicast address 233.0.0.1 and port 1234, the port would occupy the two least significant bytes of the association source. The next 4 least significant bytes will be the 4 octets of the IP address. Using this example, the association source as an integer would be 256,186,209,338,578 (E300000104D2 hexadecimal).

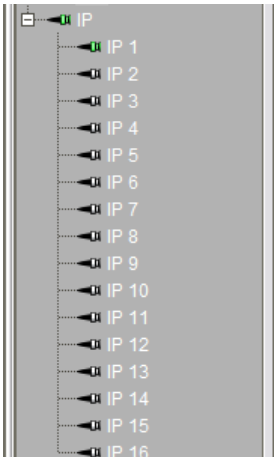


**Figure 2: IP Stream Index Example**

**Association Program:**
Specifies the transport stream program number the metadata payload is contributing to.

**Association PID:**
Specifies the intended transport stream PID value of the metadata payload.

**Inject Time:**
Specifies a time to transmit packet based on LTC time input. Inject time will be structured as follows:

| | |
|---|---|
| Reserved | 1 byte |
| Hour | 1 byte |
| Minute | 1 byte |
| Second | 1 byte |
| Frame Number | 1 byte |

Where hour has the range of 0 – 23, minute and second have the range of 0 – 59, and frame number has the range of 0 – 29. Provided inject time will be compared against incoming LTC input and when the LTC time and inject time matches MUX will insert the transport stream packets.

**NOTE:** If the user does not want to use time stamp, then all the bytes for inject time parameter should be set to 0xFF. In this case the packets will be sent out as soon as the metadata is received by MUX.

**Payload Length:**
Length of the subsequent payload, following the reserved section, in bytes. Maximum length supported is 1024 bytes

**PTS Mode:**
Specifies PTS stamping mode. Following values are possible
0 – Immediate, PTS to PCR difference will be set to minimum.
1 – Follow video's PCR to PTS difference
2 – Add fixed user specified offset. Offset can be selected using "PTS Offset" field in metadata header.
Other values – will be treated as unspecified and card will use the values specified via SNMP.

**PTS Offset:**
Offset value that will be added to PTS value of metadata packets. This is a singed value. Value is in 90 Khz ticks.

**Reserved:**
For future use.

## 3.2 Metadata TS Packet Insertion

As the metadata file is received over TCP connection, the MUX will parse the header of the packet, and subsequently generate a transport stream packet(s) using the payload of the metadata file. The MUX will then allow the transport stream packet to contribute to the service that it is destined for, based on the information provided in the metadata header. See Figure 3: MPTS to SPTS with Transport Stream Packet Insertion to Program.

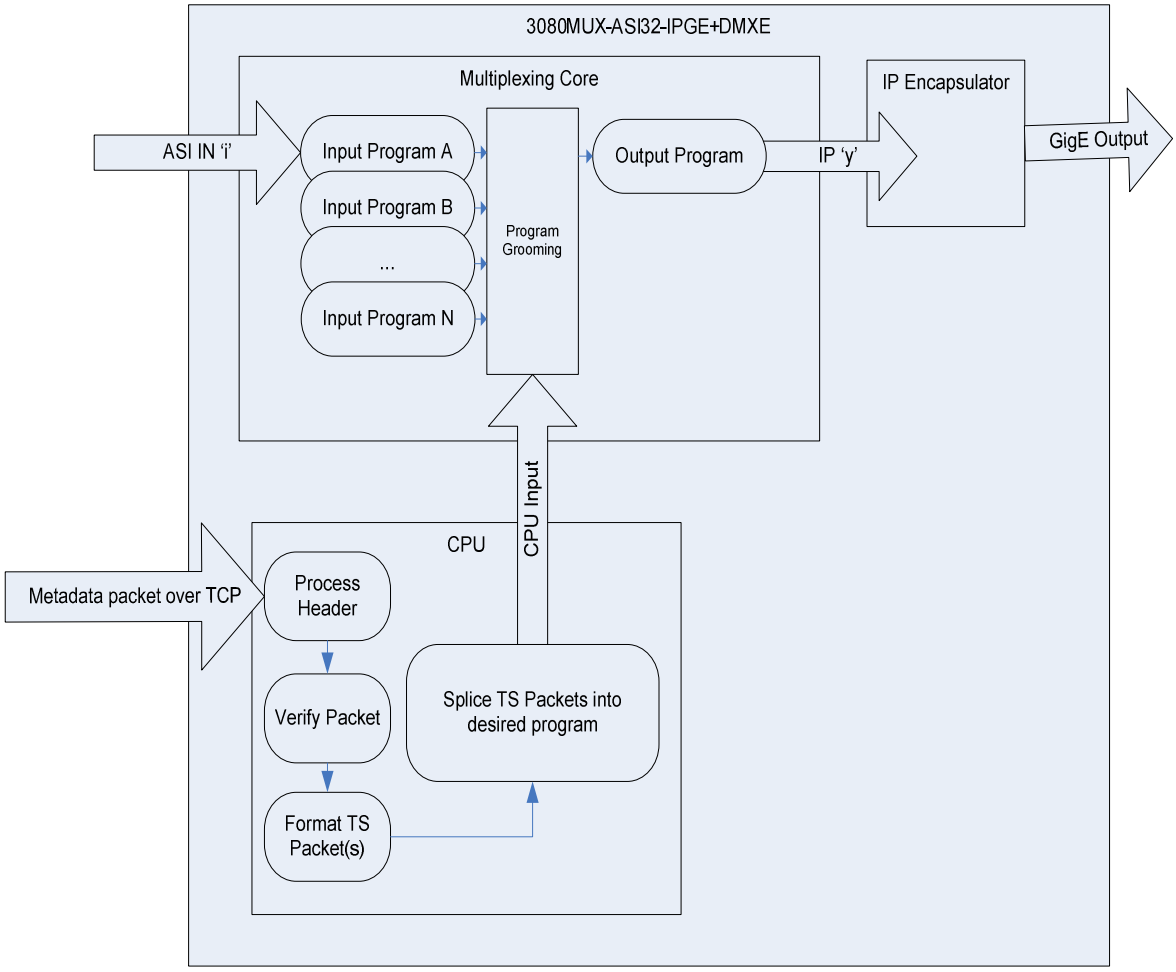



**Figure 3: MPTS to SPTS with Transport Stream Packet Insertion to Program**

# Section A

# LTC Insertion

The purpose of this section is to outline the functionality of the LTC insertion on 3080MUX-ASI32-IPGE-DMXE hardware.

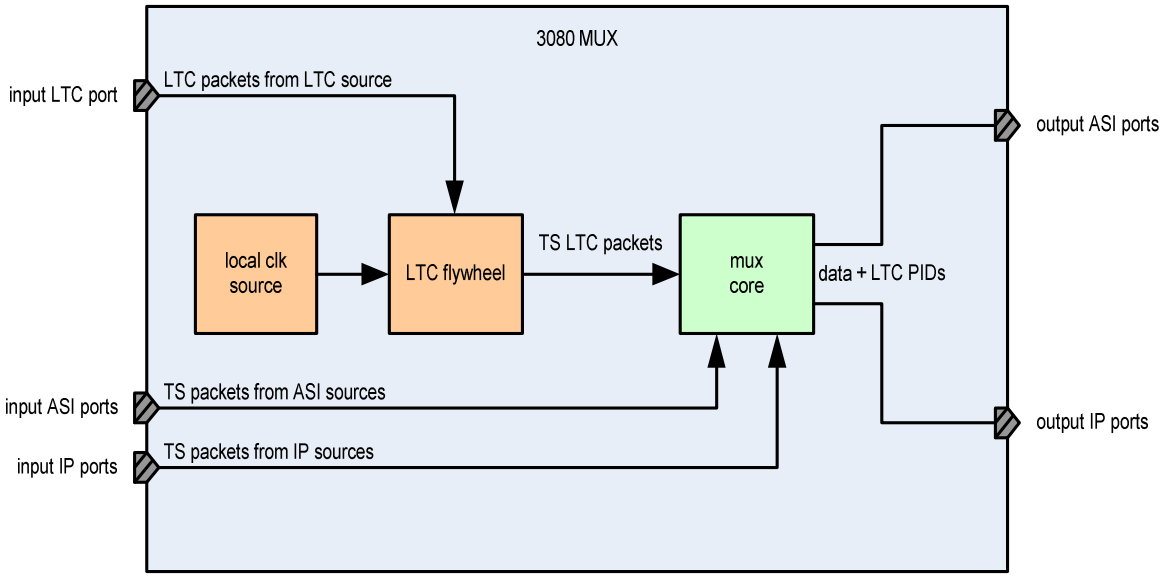
www.evertz.com

**EVERTZ MICROSYSTEMS LTD.**
***BURLINGTON, ONTARIO, CANADA***

# 1.0 Purpose

The purpose of this section is to outline the requirements for providing the 3080MUX-ASI32-IPGE-DMXE (herein referred to as MUX) with LTC input that it can then transmit as part of the output transport stream service in the form of a transport stream packet.

# 2.0 Functional Details




The operation of internal LTC logic inside the 3080 MUX is such that it receives LTC payloads at the input port from an external LTC source. It then synchronizes the timing of these LTC input packets internally and forms LTC payloads into TS packets according to MPEG transport stream specs.

The LTC TS packets may then be inserted into any outgoing port that carries any number of MPEG services on ASI or IP output ports.

An enhanced feature of the internal LTC generator is the flywheel functionality that keeps up the outgoing LTC packets, up to a user defined interval, in case of glitches on the input LTC stream.

Another enhancement is the fact that the LTC flywheel is able to synchronize itself on the input LTC stream timings without any user intervention.

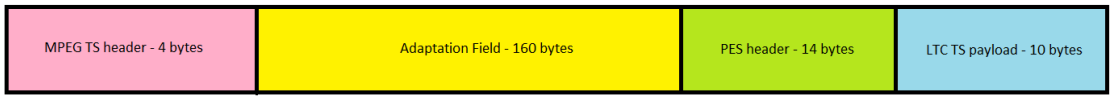
Additionally, the PID chosen to transmit the LTC TS packets is user configurable. The output LTC TS transmission may be transmitted (or inhibited) on a per-output basis.

The following frame rates are supported:

1. 23.98 fps
2. 24 fps
3. 25 fps
4. 29.97 fps
5. 29.97 fps (with Drop Frames)
6. 30 fps

## 3.0 Format of the transmitted LTC packet

The LTC TS packet has the basic MPEG TS structure:




The fields in the MPEG TS header are setup as follows:

*transport_error_indicator* – set to 1'b0

*payload_unit_start_indicator* – set to 1'b1

*transport_priority* – set to 1'b0

*PID* – set to the 13 bit user defined value that is desired for the LTC transmission

*transport_scrambling_control* – set to 2'b00

*adaptation_field_control* – set to 2'b11 to indicate adaptation field and payload

*continuity_counter* – this 4 bit field is managed by the LTC logic to make sure that there are no breaks

The adaptation field follows the structure in ISO 13818-1 standard. It includes the following fields:

*Adaptation_field_length (number of bytes in the adaptation field immediately following this byte)* – 1 byte, set to 159 decimal or 0x9F.

*discontinuity_indicator* – 1 bit, set to 0
*random_access_indicator*– 1 bit, set to 0
*elementary_stream_priority_ indicator*– 1 bit, set to 0
*PCR_flag*– 1 bit, set to 0
*OPCR_flag*– 1 bit, set to 0
*Splicing_point_flag*– 1 bit, set to 0
*Transport_private_data_flag*– 1 bit, set to 0
*Adaptation_field_extension_flag*– 1 bit, set to 0

*Stuffing_byte*– 158 bytes, set to 0xFF

The PES Header follows the PES_packet structure mentioned in ISO 13818-1 standard. The fields in the PES Header are as follows

*packet_start_code_prefix* – 3 bytes, set to '0000 0000 0000 0000 0000 0001' (0x000001).

*stream_id* – 1 byte, set to b1011 1101 (0xBD).

*PES_packet_length* – 2 bytes, set to 18 (8 bytes for PES header following PES_packet_length and 10 bytes for LTC data).

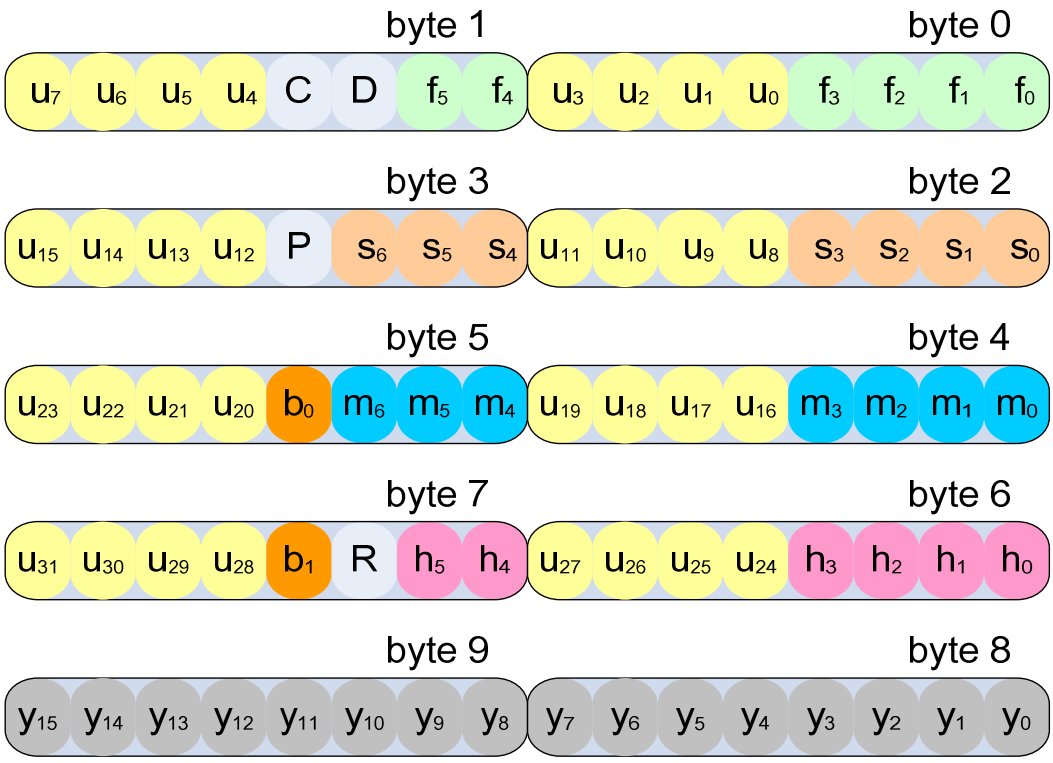
*flags* – 2 bytes, various flags in PES header.
*PES header data length* – 1 byte, set to 5.

*PTS* – 5 bytes, 33 bit PTS value muxed in as follows:

b32_30 = (PESHeader & 0x0E00000000) >> 3 ) #pts 32_30
b29_15 = ((PESHeader & 0x00FFFE0000) >> 2 ) #pts 29-15
b14_0 = ((PESHeader & 0x000000FFFE) >> 1 ) #pts 14-0
pts = b32_30 + b29_15 + b14_0

The LTC TS payload itself is always 10 bytes composed exactly as described in the LTC specs, these bytes are described next.

The following picture represents the 10 byte LTC as it is produced at the output of MUX.




Treatment of these bits by the MUX is explained next.

- **Bits $f_0$ .. $f_5$**

  These bits represent the current frame number. These are generated by the MUX and inserted into the LTC packet. They are synchronized to the input LTC signal (when present). The following expression is used to calculate the frame number based on these bits:

  frame number = $f_5 \times 20 + f_4 \times 10 + f_3 \times 8 + f_2 \times 4 + f_1 \times 2 + f_0$

- **Bit D**

This is the "Drop Frame flag" as specified in the LTC specs. This bit is copied directly from the input LTC packet without modification.

- **Bit C**

This is the "Color Framing" bit as specified in LTC specs. This bit is transferred as-is from the input LTC packet to the output.

- **Bits $s_0$ .. $s_6$**

These bits represent the current value of the “seconds” field in the timestamp. These are generated by the MUX. They are synchronized to the input LTC signal. The following expression is used to calculate the seconds field:

$$seconds = s_6 \times 40 + s_5 \times 20 + s_4 \times 10 + s_3 \times 8 + s_2 \times 4 + s_1 \times 2 + s_0$$

- **Bit P**

This is the “parity” bit as specified in the LTC specs. This bit is transferred unmodified from the input LTC to the output of the MUX.

- **Bits $m_0$ .. $m_6$**

These bits represent the current value of the "minutes" field in the timestamp. These are generated by the MUX. They are synchronized to the input LTC signal. The following expression is used to calculate the minutes field:

$$minutes = m_6 \times 40 + m_5 \times 20 + m_4 \times 10 + m_3 \times 8 + m_2 \times 4 + m_1 \times 2 + m_0$$

- **Bits $b_0$ .. $b_1$**

These are the “binary group flag” bits as specified in the LTC specs. They go through the MUX unchanged.

- **Bits $h_0$ .. $h_5$**

These bits represent the current value of the "hours" field in the timestamp. These are generated by the MUX. They are synchronized to the input LTC signal. The following expression is used to calculate the hours field:

$$hours = h_5 \times 20 + h_4 \times 10 + h_3 \times 8 + h_2 \times 4 + h_1 \times 2 + h_0$$

- **Bit R**

This is the “reserved zero bit” according to the LTC specs and it goes through the MUX without any modifications.

- **Bits $u_0$ .. $u_{31}$**

These are the 32 user defined bits. They are not manipulated by the MUX in any way and go straight through the MUX. If the input LTC goes missing, the last value found in these bits is copied to subsequent frames driven by the internal LTC flywheel.

- **Bits $y_0$ .. $y_{15}$**

These are the fixed pattern sync bits that append each LTC frame. The pattern is fixed at:

$$sync\ bits = 0xFCBF$$

These are internally generated by the MUX, not copied from the input LTC packet.

# 4.0 User configurable parameters

The LTC functionality may be configured by the following parameters through VLPRO:

- **LTC Enable**

On a per-output basis, this configures whether to transmit the LTC packets within the programs on this output or not. It has 2 possible values as shown:

- Enable: transmit LTC packets on this output
- Disable: do not transmit LTC packets for this output

- **LTC PID**

This configures the PID number on to which the LTC TS packets shall be transmitted. Possible values are 16 – 8190 while respecting the reserved PID numbers.

- **LTC Present**

This is a global read-only flag to let the user know whether the 3080 MUX is currently receiving a valid LTC input on its input LTC port.

- **LTC flywheel count**

This value is a global number indicating the number of frames for which to continue LTC transmission in case of input LTC loss. After these many frames have elapsed, the output contains a zero time stamp, i.e. 00:00:00:00 is continuously transmitted.